ARMY RESEARCH LABORATORY

The High Level Architecture Functional Class Middleware (HLAfc) Technical Introduction

by Geoffrey C. Sauerborn

ARL-TR-3146 **April 2004**

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

DESTRUCTION NOTICE—Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ARL-TR-3146 **April 2004**

The High Level Architecture Functional Class Middleware (HLAfc) Technical Introduction

Geoffrey C. Sauerborn
Weapons and Materials Research Directorate, ARL

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
April 2004	Final	October 2003 to March 2004

4. TITLE AND SUBTITLE
The High Level Architecture Functional Class Middleware (HLAfc) Technical Introduction

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Geoffrey C. Sauerborn (ARL)

5d. PROJECT NUMBER
621618H80

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
Weapons & Materials Research Directorate
Aberdeen Proving Ground, MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-3146

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
"Middleware" is a term used for a software library specifically designed to act as an agent (or a software layer that stands logically) between distinct systems (e.g., between a final user application and a targeted service such as a network communication protocol). Adding another layer of software will never increase execution speed; therefore, the benefits must outweigh this (possibly slight) penalty. Some middleware benefits over a targeted service could include portability, easing migration to other services, simplicity, software readability, maintainability, having an abstracted centralized area for error handling, data translation, "bookkeeping" tasks, and other required maintenance.

This report introduces a middleware for the U.S. Department of Defense and the Institute of Electrical and Electronics Engineers modeling and simulation high level architecture. The middleware is often described through the use of illustrations (small software examples). Yet, this description remains at a technically high point of view and is not intended as a programmer manual. The middleware's chief features are highlighted, and comparisons are made with other approaches to middleware.

15. SUBJECT TERMS
distributed simulation; lethality; simulation interface; vulnerability

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Geoffrey C. Sauerborn
Unclassified	Unclassified	Unclassified	UL	20	19b. TELEPHONE NUMBER (*Include area code*) 410-278-8657

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Contents

List of Figures

List of Tables

Acknowledgments

The author wishes to express grateful acknowledgment to MyVan H Baranoski of the U.S. Army Research Laboratory (ARL), who was this report's technical reviewer and provided some feedback during early development (while working on the ground vehicle dynamics model). Similarly, Richard Pearson, also of ARL, is thanked for his innovative dynamic system "Goodbye World" that was also used in HLAfc early tests as well as for his diligent work on the combat vehicle engineering simulation high level architecture port.

INTENTIONALLY LEFT BLANK

1. Introduction

This document describes a simulation service "middleware" that supports the Department of Defense (DoD) high level architecture (HLA). This middleware stands "in the middle" as a functional class between the HLA run time infrastructure (RTI) and an application. We refer to this middleware as the HLA functional class (HLAfc) because it provides almost all the functionality of the underlying HLA services while making HLA extremely simple to implement.

HLAfc's origins, its advantages, and how it has been used, are described. Some comparisons are made with similar services (other types of middleware architectures). This is approached from a technically high level and therefore not suitable or intended as an applications programmer manual for using HLAfc.

2. Background

HLAfc was developed by the author while he was supporting the U.S. Army's Developmental Test Command's Virtual Proving Ground (VPG) at the U.S. Aberdeen Test Center. At the time, the U.S. Army Research Laboratory (ARL) was providing an HLA component version of an Army engineering-level simulation of combat vehicles. This simulation was the combat vehicle engineering simulation (CVES), and its primary purpose was to simulate a vehicle's ballistic fire control (*1,2*). CVES's HLA representation was divided into segments (individual HLA federates) that reflected the simulation's natural internal logical organization. This meant creating separate HLA federation simulation components to represent the vehicle's hull, turret, gunner (target tracker), etc. A total of eight federates comprised the whole CVES HLA federation (*3,4*). An advantage of this approach was that one could incorporate different fire control computers, vehicle suspension systems, or other components into the system simply by switching the component in question, which would result in a "composible" architecture for fire control systems.

Though powerful in combination, the HLA RTI application program interface (API) functions are relatively elementary services (*5*). To organize these service calls into a viable HLA federate (simulation component) requires numerous accounting and other overhead operations not provided in the base RTI APIs. The purpose of HLAfc then was to implement these overhead operations (creating, joining or leaving the HLA simulation exercise, tracking subscribed simulation data, providing updates, etc.). HLAfc has been very successful in accomplishing these objectives. The result for CVES was that each of the eight simulation components could then be made to be HLA federates without anyone having to re-engineer these overhead

operations. Furthermore, because HLAfc was to implement this capability in a manner that did not presuppose the composition of the HLA federation (the object model), it therefore could be re-used to implement any future HLA federate. As we will see in section 3, writing an HLA application can now be a relatively simple process through the use of the HLAfc middleware.

With HLAfc, CVES achieved official Defense Modeling and Simulation Office (DMSO) HLA-compliance HLA certification in 21 March 2001 for all its component federates (these are named in table 1).

Table 1. DMSO-certified HLA-compliant HLAfc federates.

Certified Federate Name
VPG CVES Blue (vehicle target space position information) Platform
VPG CVES Wheel Platform
VPG CVES Platform Dynamics
VPG CVES Turret Platform
VPG CVES Red (target) Platform
VPG CVES Gun Platform
VPG CVES Hardware (fire control computer) Platform
VPG CVES Human (target tracking system) Platform

HLAfc has made other applications HLA compliant and been tested with various other experimental, as well as "production", HLA object models. Some of these are shown in table 2. In particular, the real-time platform-level federation object model (RPR-FOM) (*6*) and modeling architecture for technology and research experimentation (MATREX) (*7*) federations are large and fairly complex object models attesting to HLAfc's flexibility and robust nature.

Table 2. Examples of other HLA implementations with HLAfc.

FOM Name	Description
HelloWorld	An HLAfc version compatible with the "HelloWorld" sample program distributed with the DoD RTI versions.
GoodByeWorld	As with HelloWorld, this is a sample application. It was developed by Richard Pearson, ARL. The GoodByeWorld consists of two federates: one increasing population with time (as does HelloWorld), and a second that dynamically introduces plagues that reduce the population of the first.
GVDM	Ground Vehicle Dynamics Model. An ARL HLA-wrapped integration of DADS (vehicle dynamics simulation system) and compact terrain data base (CTDB) formatted terrain data (via ModSAF CTDB terrain libraries). HLA was later dropped from this project.
EntityBridge	A bridge between distributed interactive simulation (DIS) (*8*) and HLA created for a subset of the RPR-FOM. This bridge communicates the basic DIS PDUs: EntityState, Fire, and Detonation.
RPR-FOM	RPR-FOM is a distributed interactive simulation (DIS) protocol mapping to an HLA FOM. RPR-FOM was applied for the ARL table lookup vulnerability/lethality server. A second HLA federate was quickly generated with HLAfc to monitor mobility data from the Tank-Automotive and Armaments Command's vehicle dynamics mobility server (VDMS) during Research, Development, and Engineering Center federation experiments (2001).
MATREX V0.5	Lethality/vulnerability server (*9*)

Reducing the time needed to create an HLA-compliant application and being able to re-use a swift HLA integration capability repeatedly and consistently were the primary motivations for developing HLAfc.

3. Description

HLAfc makes it very easy for an application to become HLA compliant. Given a FOM and an operational code, an application can become HLA compliant literally by the addition of just a few lines of code. (Of course, the total lines of code depend on the number of RTI exchanges planned.) For a simple example, assume that an object model of a "car" object exists and that object has an attribute "color" as portrayed in the object class hierarchy shown in figure 1.

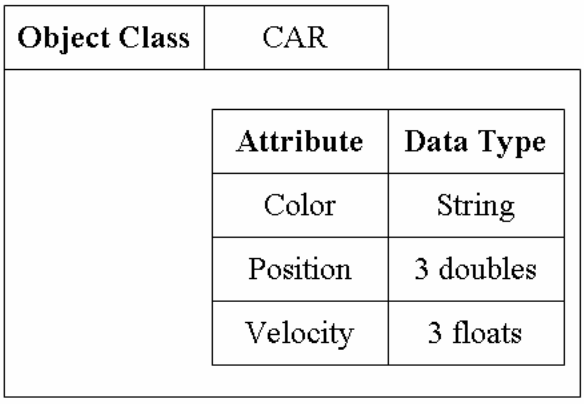

Figure 1. The "car" object model.

If the "car" object model were an HLA FOM, then the following C++ code segment portrayed in figure 2 would assign a value to "color" and publish that information to the RTI, making it available to all other HLA execution federates.

```
#include <HLAfc.hh>                  // required header file
.                                    //   dots represent ancillary C++
.                                    //   code needed to complete a
.                                    //   syntactically complete
.                                    //   program (not shown)

FOM_Object hla_fc("CarFOM.HLAfc");   // initialize HLAfc

hla_fc.rtiSetObjectAttributeValue("Car","Color", "Red"); // set value
hla_fc.rtiUpdate();                  // publish that data to the HLA federation.
```

Figure 2. HLAfc minimal C++ coding implementation example.

HLAfc executes the required HLA RTI protocol operations ("hand shaking"), data marshaling/ de-marshaling, and (eventual) "graceful" (proper) resignation from the federation execution. Of course, HLAfc can do much more than portrayed in figure 2.

4. Comparison With Other Middleware Features

Most code generators will consume an object model and produce code header information or outright classes that reflect the name space and object hierarchy for that given object model. Function calls (or methods) are "stubbed out".[1] A programmer then starts with these prototyped methods and fills them with behavior code.

HLAfc differs in this approach in that the behavior code remains in the user's original application. That is, an existing program does not have to be changed very much to communicate with other HLA applications. Instead of filling stubbed out behavior code, the user application interfaces with HLAfc to send (set) or receive (get) these exposed data when they are needed.[2]

This approach frees the user application from having to become too embedded to a particular external communications protocol (such as HLA, DIS, COM, etc.). External variable names and the objects to which they belong may be referenced by text strings that correlate to the object model components. (This was seen in figure 2 where "car" and "color" were applied during the "rtiSetObjectAttributeValue()" method invocation.) Alternatively, these object model components may be referenced by RTI handles (integers), thus providing greater efficiency and speed. (For example, integer variables that represent "car" and "color" are substituted for the text strings.) This is shown in the "rtiSetObjectAttributeValue()" invocation seen in figure 3.

Integer operations (RTI handles) have much less overhead and will therefore allow the final application to run faster. This efficiency comes at a slight administrative cost in that the user's application will be responsible for obtaining (done once) and then keeping track of those RTI handles as shown in figure 3. (In this example, the application ought to ensure that the object and attribute handle variables ["objHandle" and "atrHandle"] remain in scope in case this section of code is re-entered. Doing so will prevent our having to re-assign these handles. In a similar vein, the hla_fc object shall be instantiated only once and then used throughout the program. In fact, if hla_fc falls out of scope, the federate will resign (leave) the HLA execution.) Note the "RTI::" class data types (i.e., RTI::ObjectClassHandle and RTI::AttributeHandle) are the HLA native data types defined inside the HLA RTI distribution (*10*) and are not HLAfc-defined objects.

[1] "Stubbed out": the method's name is prototyped, but code within that method is left blank.

[2] As a convenience, HLAfc code generates classes for enumerated data types along with legal values, but their implementation in the user's application is not a requirement in order to use HLAfc.

```
   #include <HLAfc.hh>                // required header file
   .                                  //   dots represent ancillary C++
   .                                  //   code needed to complete a
   .                                  //   syntactically complete
   .                                  //   program (not shown)



   FOM_Object hla_fc("CarFOM.HLAfc"); // initialize HLAfc

RTI::ObjectClassHandle objHandle;   // will replace "Car"
RTI::AttributeHandle   atrHandle;    // will replace "Car.Color"

objHandle = hla_fc.getObjectClassHandle( "Car");
atrHandle = hla_fc.rtiGetAttributeHandle("Car", "Color");



   hla_fc.rtiSetObjectAttributeValue( objHandle, atrHandle, "Red"); // set


   hla_fc.rtiUpdate();                // publish that data to the HLA federation.
```

Figure 3. Example of applying RTI object handles to HLAfc updates.

4.1 Condensed List of Other Features

Table 3 provides a few of HLAfc's highlighted features.

Table 3. HLAfc key features.

Feature	Description
Simplicity of use	HLAfc is intuitive. In its basic form, HLAfc distills to simple "set" (or "get") API calls to assign (or retrieve) HLA data.
Develop-ment speed	As figure 2 demonstrated, it takes little time for an application programmer to comply with the HLA specification.
FOM persistence	Once the middleware has included a FOM, it stays resident. This means that there is no need to recompile when one is switching from one FOM to another, if such a need arises.
Configura-tion file (.HLAfc file)	A single configuration file determines which federation is to be subscribed to (federation name), the federate's name, federation (.fed file), as well as subscribed objects (those HLA objects that will be published or subscribed to by the federate application). • Most of this configuration file is auto-generated when the FOM is "read" (or consumed) by HLAfc. (The consumption process is not within the scope of this document.)
Error handling	Generous application of useful error handling such as • Attempts to publish or subscribe to objects not in the FOM. • FOM type checking (write): e.g., attempts to send a random data stream that would overflow a known object length. (This type of checking is a compilation option that may be turned off; default is ON.) • FOM type checks (read): received data that would overflow an object are truncated and noted as errors. • Warnings for truncated objects: if a data structure of a known length is received with a lesser size. • Many more types of errors. • Because of HLAfc's data length checking, one may find many types of FOM interface errors over the whole HLA federation execution simply by subscribing to all HLA objects and

	listening without further action necessary. When a rogue federate overflows (or truncates) any object's data field, HLAfc will immediately report this and do its best to identify the offending federate application.
Configu-rable Logging	Automatically log any HLA incoming or outgoing traffic (see the rtiVerboseLogIO() method). (Current format is intended for manual review and not designed as a log file for a playback operation.)
Data Com-munications flexibility	Once data are passed to HLAfc, they handle any "on-the-wire" standards (such as network byte ordering). XDR (*11*) data representation is the default (but may be turned off with a compile option). The numeric data-type network byte-ordering subset of XDR will remain.
More Debugging utilities	Trace calls to internal HLAfc methods, tunable verbosity of the data trace and debug output (via rtiVerbose(), rtiVerboseTrace() methods) (though many of these will only be useful to an experienced HLA programmer).

Maximum flexibility	An experienced HLA programmer can go beyond the basics if needed. Callbacks (for object discovery), and other resources that tap many of the most used RTI services are available and seen in appendix A. If ***any*** HLA service is needed, one may access it by obtaining and applying the Federate Ambassador and RTI Ambassador pointers to the joined federation's execution:
	`extern rtiFC_FederateAmbassador fedAmb;//fed<--RTI`
	`extern RTI::RTIambassador rtiAmb; //fed -->RTI`
	fedAmb is used by the RTI to communicate to the user's applications, where rtiAmb is used by the application program to send data to the RTI. (See reference 10, appendices A and B)

4.2 System Requirements

4.2.1 Language and Operation System

HLAfc is a C++ compiled library. (There are currently no JAVA[3] or other language bindings.) Its currently maintained version is compiled with the GNU (Gnu's Not Unix) C++ 3.2.2 compiler on RedHat Linux 9.0. It has worked in the past with the Silicon Graphics, Inc. (SGI) MIPS[3]pro C++ compiler, under IRIX (an SGI UNIX-like operating system).

4.2.2 HLA RTI Support

HLAfc has also been tested to support

- RTI-1.3NGjvbV3b1 (RTI NG-PRO from Virtual Technology Corporation incorporated into the joint virtual battlespace and MATREX federation).
- RTI 2.0.1ngc, from MaK[3] Technologies
- DMSO RTI-NG1.3v6[4]

4.3 Recommended Programming Experience

As figures 2 and 3 demonstrated, programs can be straightforward. To access more powerful features, more programming experience will be needed. However, any C++ programmer with a

[3]Not an acronym.

[4]HLAfc supports all versions of the DMSO-distributed HLA RTI distributions but is maintained under DMSO RTI-NG1.3v6.

few months' experience should be able to very quickly implement at least a basic HLA program using HLAfc.

4.4 Complex Data Types

HLAfc is able to pass and receive complex data structures. Notice in figures 2 and 3 that a string value was published ("Red"). HLAfc knew to expect a string because the data element Car.Color was defined as a string data type in the federation object model depicted in figure 1. As previously mentioned, HLAfc consumes the HLA standard object model template (OMT) stored in the standard HLA 1.3 OMT data interchange format (DIF). The OMT defines all data types including complex (user-defined) types. Complex data structures also may be passed to or read from the RTI. However, the user application must first serialize the data. This process removes any byte padding or other alignments. To continue with our example, suppose that the "car" object's attributes were defined in one complex data structure containing the same types seen in figure 1 (a variable length null-terminated string, an array of three doubles, and an array of three floats, in that order). Then the application must first store these three values into a buffer (the null-terminated string, followed immediately by the three 32-bit floating point array values, and finally the three 64-bit double precision floating point values). Once these are placed (in this order) into the buffer, a pointer to that buffer is passed as the third parameter to the rtiSetObjectAttributeValue() call. Of course, the first two parameters would indicate the object and attribute name of this hypothetical complex data type. HLAfc will recognize that this object model component name is associated with a complex data type and will internally marshal the data for any "on-the-wire" standards (such as converting the null-terminated "C" string into an XDR string, byte ordering the arrays of doubles and floats, etc.).

On the receiver end, HLAfc will reverse any "on-the-wire" standards and return a buffer containing the serialized data in host computer format (via the rtiGetObjectAttributeValue() method call). The receiving user application must then extract these serialized data from the returned buffer. HLAfc can service any complex data type, but there is currently no provision for data structures of unknown length (except for variable length strings).

5. Summary

HLAfc has been introduced. Motivations for having a middleware such as HLAfc, its particular origins, and how it has already been applied were provided. Among HLAfc's chief advantages are the speed and simplicity at which a useful HLA application-level program may be generated. HLAfc abstracts data apart from the code generation process and applies a series of "set" and "get" operations to interface exposed HLA data. This is a different approach from other code-generation-based middleware architectures.

While this is not a programmer's manual, a simple use case has been provided to allow technical evaluators a first look into how this middleware can be applied at the application level and to make some preliminary evaluations.

6. Conclusion

HLAfc's demonstrated ability to service a broad range of HLA federation object models authenticates its robust nature. Object model data abstraction provides a highly flexible and simplified interface based on "set" and "get" APIs. This approach can allow extreme speed in HLA compliance.

Data abstraction from the code also allows powerful flexibility. For example, if desired, a simulation could switch to other object models and yet have its code base remain basically unchanged (by the object names being changed at the application layer, provided the new object name maps usage and data type between the two FOMs).

Despite its success, HLAfc still has some limitations, the greatest of which being there is currently no provision for data structures of unknown length. Less elegant (than being imbedded within HLAfc) "work-arounds" are available, but these require preconceived knowledge of how the FOM intends to communicate the actual length of the data structures sent and received. Therefore, the solution in this case remains at the application level (negating the reason for having a middleware for such structures). However, HLAfc does provide "hooks" into the native HLA interface for such circumstances.

7. References

1. Corcoran, P. E. *Gunner Tracking Models for the BFVS-A3 Combat Vehicle Engineering Simulation*; ARL-TR-2588; U.S. Army Research Laboratory: Aberdeen Proving Ground, Maryland, November 2001.

2. Corcoran, P. E. *Gunner Tracking Models for the M1A1 Combat Vehicle Engineering Simulation*; ARL-TR-1984, U.S. Army Research Laboratory: Aberdeen Proving Ground, Maryland, May 1999.

3. Pearson, R. J.; Sauerborn, G. C.; Kenrick C. Lessons Learned Migrating Legacy Engineering Models to the HLA (CVES a Case Study), Simulation Interoperability Standards Organization (SISO), Spring 1999 Simulation Interoperability Workshop (SIW) Workshop Papers", March 1999.

4. Pearson, R. J.; Sauerborn, G. C., Kenrick C. Migration of Legacy Engineering Models to a High Level Architecture (HLA) Simulation, International Test & Evaluation Association, published in *Proceedings of the 3rd Modeling and Simulation Workshop - Establishing Seamless Distributed and Integrated Solutions to Real-World Challenges*, December 1998.

5. IEEE Computer Society, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) Federate Interface Specification; IEEE Std 1516.1-2000, Institute of Electrical and Electronics Engineers (IEEE), 21 September 2000.

6. Fischer, J.; Case, R.; Bertin, R. Eds. Guidance, Rationale, and Interoperability Manual for the Real-time Platform Reference Federation Object Model (RPR FOM), Version 2.0D14v2, SISO, 11 March 2002.

7. Joint Precision Strike Demonstration Office (JPSD), Modeling Architecture for Technology and Research EXperimentation (MATREX) System Architecture Description, Coordination Draft, V0.8, JPSD, Ft. Belvoir, Virginia, May 15, 2003.

8. IEEE Computer Society, IEEE Standard for Distributed Interactive Simulation; IEEE Stds 1278.1, 1278.2, 1278.4, 1278.5; IEEE 1995, 1996, 1997, 1998.

9. Sauerborn, G.C.; Christy, T.S. *Lethality/Vulnerability Server Functional Description and Interface Control Document for MATREX V0.5*; ARL-MR-0582; U.S. Army Research Laboratory: Aberdeen Proving Ground, MD, March 2004.

10. Department of Defense, High Level Architecture Run-Time Infrastructure Programmer's Guide, 1.3 Version 4, DoD Modeling and Simulation Office, 21 September 1998.

11. Network Working Group, XDR: External Data Representation Standard, Sun Microsystems, Request for Comments (RFC-1014), June 1987.

INTENTIONALLY LEFT BLANK

Appendix A. FOM_Object Class Member List

This appendix contains a condensed listing of the names of most HLA APIs services available through the FOM_Object class (HLAfc's interface class). The list is provided in alphabetical order for informational purposes as opposed to programming instructions. As such, and keeping with the intent of this text, detailed API explanations, returned values, exceptions to handling and proper usage are not provided in this condensed listing. Most of the internally used (private and static) methods are listed first (the majority of whose names start with the underscore "_"). In practice, user applications will never invoke these methods. A typical application will confine itself to those methods whose names start with "rti" in lowercase (e.g., rtiUpate() , rtiSetObjectAttributeValue(), rtiGetObjectAttributeValue(), etc.).

Other items that will not be needed by most applications are the SOM_Object and SOM_Interaction classes. These are used internally to handle HLA attributes and parameters and are therefore not documented in this listing.

The other notable set of classes not documented are native HLA RTI implementation classes (the "RTI::" classes, i.e., RTI::ObjectClassHandle and RTI::AttributeHandle). These are the HLA native data types defined inside the HLA RTI distribution (*5, 10*) and are not HLAfc-defined classes.

This is the complete list of members for **FOM_Object**, including all inherited members.

Method or object name	Exposure or compile option
__create_new_fom_object(RTI::ObjectClassHandle theObjectClass)	[static]
__create_new_fom_object(RTI::ObjectClassHandle theObjectClass, SOM_Object **newSomPtr)	[static]
__getFederateLocalTime_non_RTI_time()	
__getsim(const char *nm)	[inline, private]
__set_som_object_as_local_object(SOM_Object *newSom)	[static]
__set_som_object_as_remote_object(SOM_Object *newSom)	[static]
_currentTime	[private]
_Federation_RTI_FedFileName	[private]
_FederationName	[private]
_get_fom_of_exactly_one_atrib_Remote(const char *ObjName, const char *AttrbName)	[private]
_get_sim_of_exactly_one_param(RTI::InteractionClassHandle ParamHandle, SOM_Interaction **asim, const char **errmsg)	[private]
_get_sim_of_exactly_one_param(const char *Pname, SOM_Interaction **asim, const char **errmsg)	[private]
_get_som_of_exactly_one_atrib(const char *AttrbName, SOM_Object **asom, const char **errmsg)	[private]
_get_som_of_exactly_one_atrib(RTI::AttributeHandle AttrbHandle, SOM_Object **asom, const char **errmsg)	[private]
_get_som_of_exactly_one_atrib_Remote(const char *ObjName, const char *AttrbName)	[private]
_getActiveFOM_Object()	[inline, private]
_HLAfc_ObjectMode_Name	[private]
_HlaObjNameCmp(const char *a, const char *b)	[private]
_lookaheadTime	[private]
_rtiFOM_Object_Internal_numberOfInternalSOMs()	[inline]
_rtiFOM_Object_Internal_SOM_interateNext(int i)	[inline]
_rtiFOM_Object_Internal_soms_getbyclass_Remote(RTI::ObjectClassHandle classHandle)	[inline]
_rtiProvideAllObjectInstacesRemote(bool getAll, RTI::ObjectHandle getJustThisType)	[private]
_rtiUtilNameIsInList(const char *objNm, const char *object_names[])	
_rtiWaitForAtributeUpdatesStart(const char *ClsNm, const char *attibnames[], float small_tick_time_slice)	
_sims_add(SOM_Interaction *n)	[inline]
_sims_create()	[inline]
_sims_get(const char *nm)	[inline]
_sims_getbyclass(RTI::InteractionClassHandle classHandle)	[inline]
_sims_geti(int i)	[inline]
_sims_getParam(RTI::ParameterHandle ph)	[inline]
_sims_getParam(const char *pname)	[inline]
_sims_size()	[inline]
_soms_add(SOM_Object *n)	[inline, private]

_soms_create()	[inline, private]
_soms_delete(RTI::ObjectHandle instanceID)	[inline, private]
_soms_delete_all()	[inline, private]
_soms_get(const char *nm)	[inline, private]
_soms_get(RTI::ObjectHandle Objeect_Instance)	[inline, private]
_soms_getAtrib(const char *name)	[inline, private]
_soms_getAtrib(RTI::AttributeHandle ah)	[inline, private]
_soms_getbyclass(RTI::ObjectClassHandle classHandle)	[inline, private]
_soms_getbyclass_Remote(RTI::ObjectClassHandle classHandle)	[inline, private]
_soms_getbyName_Local(const char *nm)	[inline, private]
_soms_getbyName_Remote(const char *nm)	[inline, private]
_soms_getbyName_Template(const char *nm)	[inline, private]
_soms_geti(int i)	[inline, private]
_soms_size()	[inline, private]
add_to_list(FOM_Object *it)	[inline, private]
allobjects	[private, static]
ascii()	
ascii_Interaction(const char *interactionName)	
ascii_Interactions()	
federate_name	[private]
federateId	[private]
FOM_Object()	
FOM_Object(const char *SOMfileAtrributes)	
fom_object_zero()	[private]
getFederatation_RTI_FedFileName()	
getFederateID()	[inline]
getFederateName()	[inline]
getFederationName()	[inline]
getFOM_Object(RTI::ObjectClassHandle theObjectClass)	
getFOM_ObjectInstance(RTI::ObjectHandle object_id)	
getHLAfcObjectModelName()	[inline]
getObjectAttributeClassHandle(const char *AttribName)	
getObjectAttributeName(RTI::AttributeHandle objID)	
getObjectAttributeName(RTI::ObjectClassHandle objID, RTI::AttributeHandle atrID)	
getObjectClassHandle(const char *ObjectClassName)	
getOwningFederate(RTI::ObjectHandle instanceID)	
getSOM_Interaction(RTI::InteractionClassHandle thisClass)	
getSOM_InteractionReference(const char *nm)	[inline]
getSOM_ObjectReference(RTI::ObjectClassHandle thisClass)	
getSomObject(RTI::ObjectHandle object_id)	
init_DoesNextTokenStartAnInteractionClass(FILE *)	[private]
init_DoesNextTokenStartAnObjectClass(FILE *)	[private]

isRemoteObject	
name()	[inline]
object_count	[private, static]
printAllKnownFOMObjects()	
rtiCreateAndRegisterObjectInstance(const char *ObjectClassName)	
rtiDeleteFederate(RTI::ObjectHandle instanceID)	[inline]
rtiGetAttributeHandle(const char *ClassName, const char *AttributeName)	
rtiGetInteractionClassHandle(const char *theInteractionClassName)	
rtiGetInteractionClassName(RTI::InteractionClassHandle theInteraction)	
rtiGetInteractionHandle(const char *ClassName)	
rtiGetInteractionParameterValue(const char *paramName)	
rtiGetInteractionParameterValue(const char *InteractionClassName, const char *parameterName)	
rtiGetInteractionParameterValue(const char *parameterName, int *len)	
rtiGetInteractionParameterValue(const char *InteractionClassName, const char *parameterName, int *len)	
rtiGetObjectAttributeCardinality(const char *objName, const char *atribName)	
rtiGetObjectAttributeValue(const char *atribName)	
rtiGetObjectAttributeValue(const char *atribName, int *length)	
rtiGetObjectAttributeValue(const char *objName, const char *atribName)	
rtiGetObjectAttributeValue(const char *objName, const char *atribName, int *length)	
rtiGetObjectAttributeValue(RTI::ObjectHandle instanceID, RTI::AttributeHandle Ah)	
rtiGetObjectAttributeValue(RTI::ObjectHandle, const char *AttributeName)	
rtiGetObjectAttributeValue_Local(const char *atribName)	
rtiGetObjectAttributeValue_Local(const char *atribName, int *length)	
rtiGetObjectAttributeValue_Local(const char *objName, const char *atribName)	
rtiGetObjectAttributeValue_Local(const char *objName, const char *atribName, int *length)	
rtiGetParameterHandleHandle(const char *ClassName, const char *ParameterName)	
rtiJoinFederation()	
rtiNumberOfObjectInstaces()	
rtiProvideAllObjectInstacesRemote()	
rtiProvideAllObjectInstacesRemote(RTI::ObjectClassHandle)	
rtiProvideAllObjectInstacesRemote(const char *classname)	
rtiRegisterObjectInstance()	
rtiRemoteObjectClassHasJoined(const char *className)	
rtiRemoteObjectClassHasPublished(const char *className)	
rtiRequestAllAttributeValueUpdates()	
rtiResignFromFederation()	
rtiResume_RTI_Ticks()	
rtiSendInteraction(const char *InteractionClassName)	
rtiSetInteractionParameterValue(const char *paramName, const void *data)	
rtiSetInteractionParameterValue(const char *iClassName, const char *paramName, const void *data)	

rtiSetInteractionParameterValue(const char *iClassName, const void *data, int len)	
rtiSetInteractionParameterValue(RTI::ParameterHandle a, const void *data)	
rtiSetInteractionParameterValue(RTI::ParameterHandle a, const void *data, int len)	
rtiSetInteractionParameterValue(RTI::InteractionClassHandle ih, RTI::ParameterHandle a, const void *dataArg)	
rtiSetInteractionParameterValue(RTI::InteractionClassHandle ih, RTI::ParameterHandle a, const void *dataArg, int length)	
rtiSetObjectAttributeValue(const char *atribName, const void *data)	
rtiSetObjectAttributeValue(const char *atribName, const void *data, int len)	
rtiSetObjectAttributeValue(RTI::AttributeHandle a, const void *data)	
rtiSetObjectAttributeValue(RTI::AttributeHandle a, const void *data, int len)	
rtiSetObjectAttributeValue(RTI::ObjectHandle ObjectInstance, RTI::AttributeHandle a, const void *data)	
rtiSetObjectAttributeValue(RTI::ObjectHandle ObjectInstance, RTI::AttributeHandle a, const void *data, int len)	
rtiSetObjectAttributeValue(const char *ObjClassName, const char *AttrbName, const void *dataArg)	
rtiSetObjectAttributeValue(const char *ObjClassName, const char *AttrbName, const void *dataArg, int len)	
rtiSetObjectClassHandle()	
rtiSuspend_RTI_Ticks()	
rtiSyncAllHavePublished(const char *object_names[])	
rtiSyncAllHavePublished(const char *object_names[], double smallsteptime)	
rtiSyncLoopWaitForJoin(const char *object_names[], double smallsteptime)	
rtiTick(double timeStep)	
rtiTick()	
rtiTimeConstrained()	
rtiTimeGetFederationTime()	
rtiTimeGetLookaheadTime()	[inline]
rtiTimeIncrementTime(double timeStep)	
rtiTimeRegulated()	
rtiTimeSetLookaheadTime(double t)	
rtiTimeSetTimeConstrained(bool turn_on)	
rtiTimeSetTimeRegulating(bool turn_on)	
rtiTimeSetTimeRegulating(bool turn_on, double LookaheadTime)	
rtiTimesSet(const char *interactionClassName, const char *paramName)	
rtiTimesSet(const char *InteractionClassName)	
rtiUpdate()	
rtiUpdate(RTI::ObjectHandle)	
rtiUpdatesReceived(const char *SomObjectName)	[inline]
rtiUpdatesReceived(const char *SomObjectName, const char *atrbNm)	[inline]
rtiUpdatesSent(const char *SomObjectName)	[inline]
rtiUpdatesSent(const char *SomObjectName, const char *atrbNm)	[inline]
rtiVerbose(bool Io)	[inline]

rtiVerbose()	[inline]
rtiVerboseLogIO(bool Io)	[inline]
rtiVerboseTrace(bool Io)	[inline]
setFederateID(int id)	[inline]
setFederateName(const char *str)	[inline]
setFederation_RTI_FedFileName(const char *FileName)	
setFederationName(const char *name)	
setHLAfcObjectModelName(const char *str)	[inline]
sims	[static]
soms	
~FOM_Object()	

NO. OF COPIES ORGANIZATION

* ADMINISTRATOR
DEFENSE TECHNICAL INFO CTR
ATTN DTIC OCA
8725 JOHN J KINGMAN RD STE 0944
FT BELVOIR VA 22060-6218
*pdf file only

1 DIRECTOR
US ARMY RSCH LABORATORY
ATTN AMSRD ARL CI IS R REC MGMT
2800 POWDER MILL RD
ADELPHI MD 20783-1197

1 DIRECTOR
US ARMY RSCH LABORATORY
ATTN AMSRD ARL CI OK TECH LIB
2800 POWDER MILL RD
ADELPHI MD 20783-1197

1 AMCOM MRDEC
ATTN AMSMI RD W C MCCORKLE
REDSTONE ARSENAL AL 35898-5240

1 HICKS AND ASSOCIATES
ATTN G SINGLEY III
1710 GOODRICH DR STE 1300
MCLEAN VA 22102

1 CECOM
SP & TERRESTRIAL COMMCTN DIV
ATTN AMSEL RD ST MC M H SOICHER
FT MONMOUTH NJ 07703-5203

1 US ARMY INFO SYS ENGRG CMND
ATTN ASQB OTD F JENIA
FT HUACHUCA AZ 85613-5300

1 US ARMY NATICK RDEC
ACTING TECHNICAL DIR
ATTN SSCNC T P BRANDLER
NATICK MA 01760-5002

1 US ARMY RESEARCH OFC
4300 S MIAMI BLVD
RSCH TRIANGLE PARK NC 27709

1 US ARMY SIMULATION TRAIN & INSTRMNTN CMD
ATTN J STAHL
12350 RESEARCH PARKWAY
ORLANDO FL 32826-3726

NO. OF COPIES ORGANIZATION

1 US ARMY TANK-AUTOMOTIVE & ARMAMENTS CMD
ATTN AMSTA AR TD M FISETTE
BLDG 1
PICATINNY ARSENAL NJ 07806-5000

1 US ARMY TANK-AUTOMOTIVE CMD
RD&E CTR
ATTN AMSTA TA J CHAPIN
WARREN MI 48397-5000

1 US ARMY TRAINING & DOCTRINE CMD
BATTLE LAB INTEGRATION & TECH DIR
ATTN ATCD B J A KLEVECZ
FT MONROE VA 23651-5850

1 CDR US ARMY AVIATION RDEC
CHIEF CREW ST R7D N BUCHER
MS 243-4
AMES RESEARCH CTR
MOFFETT FIELD CA 94035

1 ITT INDUSTRIES
ATTN M O'CONNOR
600 BLVD SOUTH STE 208
HUNTSVILLE AL 35802

1 DIR US ARL
ATTN AMSRD ARL SL EP G MAREZ
WHITE SANDS MISSILE RANGE NM 88002

1 DIR US ARMY TRAC
ATTN ATRC WE L SOUTHARD
WHITE SANDS MISSILE RANGE NM 88002

3 DIR US ARMY TRAC
ATTN ATRC WEC J AGUILAR
C DENNY D DURDA
WHITE SANDS MISSILE RANGE NM 88002

3 CDR TARDEC
ATTN AMSTA TR D M/S 207
FSCS R HALLE G SIMON
WARREN MI 48397-5000

3 CDR ARDEC
ATTN AMSTA AR FSS J CHU
D MILLER B DAVIS
PICATINNY ARSENAL NJ 07806-5000

NO. OF
COPIES ORGANIZATION

2 JOINT VIRTUAL BATTLESPACE
ATTN J McDONALD J GARCIA
10401 TOTTEN ROAD
BLDG 399 SUITE 325
FT BELVOIR VA 22060-5823

1 US SBCCOM
NATICK SOLDIER CTR
ATTN AMSSB RSS MA (N) D TUCKER
KANSAS STREET
NATICK MA 01760-5020

1 HQ OPERATIONAL TEST CTR
ATTN CSTE OTC MA S J HAMILL
BLDG 91012
FT HOOD TX 76544-5068

1 SANDIA NATIONAL LABORATORIES
ATTN: M J McDONALD
PO BOX 5800 MS 1004
ALBUQUERQUE NM 87185-1004

ABERDEEN PROVING GROUND

1 DIRECTOR
US ARMY RSCH LABORATORY
ATTN AMSRD ARL CI OK (TECH LIB)
BLDG 4600

1 DIR AMSAA
ATTN D JOHNSON
BLDG 248

2 DIR AMSAA
ATTN B BRADLEY A WONG
BLDG 367

3 DIR AMSAA
ATTN D HODGE P NORMAN
K STEINER
BLDG 392

5 US ARMY RESEARCH LABORATORY
ATTN AMSRD ARL WM BF
S WILKERSON
G SAUERBORN (4)
BLDG 390

6 US ARMY RESEARCH LABORATORY
ATTN AMSRD ARL SL BE L BUTLER
R BOWERS C KENNEDY
J ANDERSON T CHRISTY
E GREENWALD
BLDG 238

NO. OF
COPIES ORGANIZATION

4 US ARMY RESEARCH LABORATORY
ATTN AMSRD ARL SL BB R SANDMEYER
P TANENBAUM B WARD
W WINNER
BLDG 328

1 US ARMY RESEARCH LABORATORY
ATTN AMSRD ARL SL BE L ROACH
BLDG 328

3 US ARMY RESEARCH LABORATORY
ATTN AMSRD ARL CI CT G MOSS
M THOMAS P JONES
BLDG 321

1 US ARMY RESEARCH LABORATORY
ATTN AMSRD ARL CI CT F BRUNDICK
BLDG 1116A

1 DIR USARL
AMSRD ARL WM W J SMITH
BLDG 4600

1 DIR USARL
AMSRD ARL WM BA D LYON
BLDG 4600